Quantum Information Processing (2024) 23:61
https://doi.org/10.1007/s11128-023-04245-1

Quantum algorithms to compute the neighbour list of *N*-body simulations

E. F. Combarro[1] · I. F. Rúa[2] · F. Orts[3] · G. Ortega[3] · A. M. Puertas[4] · E. M. Garzón[3]

Received: 1 March 2022 / Accepted: 28 December 2023

Abstract
One of the strategies to reduce the complexity of N-body simulations is the computation of the neighbour list. However, this list needs to be updated from time to time, with a high computational cost. This paper focuses on the use of quantum computing to accelerate such a computation. Our proposal is based on a well-known oracular quantum algorithm (Grover). We introduce an efficient quantum circuit to build the oracle that marks pairs of closed bodies, and we provide three novel algorithms to calculate the neighbour list under several hypotheses which take into account a-priori information of the system. We also describe a decision methodology for the actual use of the proposed quantum algorithms. The performance of the algorithms is tested with a statistical simulation of the oracle, where a fixed number of pairs of bodies are set as neighbours. A statistical analysis of the number of oracle queries is carried out. The results obtained with our simulations indicate that when the density of bodies is low, our algorithms clearly outperform the best classical algorithm in terms of oracle queries.

Keywords Quantum computing · Quantum algorithm · Neighbour list · N-body simulations

✉ E. F. Combarro
efernandezca@uniovi.es

1 Computer Science Department, University of Oviedo, Oviedo, Spain

2 Mathematics Department, University of Oviedo, Oviedo, Spain

3 Informatics Department, University of Almería, Almería, Spain

4 Department of Applied Physics, University of Almería, Almería, Spain

Published online: 13 February 2024

1 Introduction

The N-body problem is widely used in simulations in a large variety of fields, from material science, statistical physics, to astrophysics [1–3]. However, the high computational load of N-body simulations is well-known. When the number of particles, N, is not too large, the interactions can be computed by a brute-force approach, with complexity order $O(N^2)$ [1, 2, 4]. Nevertheless, when N increases, it is necessary to reduce the complexity.

Barnes & Hut defined a hierarchical tree cells scheme to locate the particles and an algorithm to compute the interactions with a complexity of $O(Nlog(N))$. It is widely applied to a large number of long-range interactions ranging from stellar dynamical applications [5] to material science or molecular dynamics [1]. Moreover, an adaption of Barnes & Hut' scheme has also been simplified for the approximate computation of long-range forces between mutually interacting bodies with a complexity of $O(N)$ [6].

In the context of short-range interactions, the main approach to get a complexity of $O(N)$ is to define a neighbour list, where the interactions are only computed among neighbour particles. However, the neighbour list has to be updated after several time steps, and its complexity is $O(N^2)$. The frequency of such computation can be reduced if the neighbourhood radius is optimized, or when a cell structure can be used to box the particles [2, 7].

Our interest is the acceleration of simulations related to N-body systems with short-range interactions by the fast computation of neighbour lists. This problem becomes relevant when a suitable cell structure of the data cannot be found [8], for instance, in systems where the range of the interaction is comparable to the system size, such as phase equilibria or out-of-equilibrium soft-matter systems [9]. Particularly in suspensions of macromolecules or colloids, the interaction among the particles is of a much shorter range than the radius or typical length, but larger than the solvent molecules. In these cases, neighbour lists are a standard solution to the computation of the forces or the energy, although some alternative techniques can specifically devised with better performance for particular cases. However, the complexity is still $O(N^2)$ in the worst case, e.g. extremely dense systems, or fast microscopic dynamics or driven systems, where frequent updates of the neighbour list are necessary.

Quantum computing [10] can be considered as a strategy to predictably accelerate these computationally expensive simulations. Quantum computing relies on the basic quantum principles of superposition and entanglement, which make it suitable for accelerating parallel and distributed applications and also for improving networks and communications.

Previous works exploit the quantum parallelism in many-body system simulations based on adiabatic quantum computation [11–13]. In contrast, this paper addresses the N-body simulations considering quantum circuit algorithms to accelerate the computation of neighbour lists. It is designed using Grover's algorithm, the main oracular quantum search algorithm [10].

The aim of this paper is twofold. Firstly, to propose several comprehensive solutions to the computation of the neighbour list with quantum computing under different alternative hypothesis. The algorithms proposed here are tested with a simplified oracle,

where a fixed number of pairs of particles are set as neighbours. The circuits obtained from this study are freely available at https://github.com/2forts/qsec. Secondly, to set a decision methodology for the actual use of the proposed quantum algorithms. And, additionally, to set a design methodology for the development of quantum algorithms, taking into account a comprehensive design that supplies both algorithms and related circuits.

The manuscript is organized as follows. Section 2 is devoted to describing the three proposed quantum algorithms for finding pairs of close particles and the selection criteria. Furthermore, details about the oracle design as a reversible quantum circuit are discussed. In Sect. 3, statistical simulations to test the proposed algorithms with a simplification of the oracle are carried out. Finally, the conclusions are presented.

2 Quantum algorithms for finding pairs of close particles

In this section, we propose three quantum algorithms that can be used to find all the pairs of particles that are closer than a given threshold distance. They are all based on Grover's algorithm, a quantum method that performs a search through an unstructured space, achieving a quadratic speed-up with respect to classic search algorithms. Among other quantum properties, Grover's algorithm is based on the concepts of superposition and quantum parallelism to compute several evaluations of a function as one [14]. The algorithm obtains a solution with a certain probability, requiring a minimum of iterations of the algorithm to get the solution with the desired probability. The estimation of the necessary number of iterations is one of the most important parts of the algorithm.

Grover's algorithm needs a black box oracle O as an input. This oracle has to check if a value x is (or not) a solution to the search problem. Therefore, to apply Grover's algorithms to a real problem, it is necessary to have an oracle with the capacity to recognize if a given value is a valid solution to that problem.

Thus, we will assume, as it is customary in this kind of problem [10, 14], that we are given a quantum circuit implementing an oracle O such that

$$O(|x\rangle|0\rangle) = \begin{cases} |x\rangle|1\rangle & \text{if } x \text{ satisfies certain conditions} \\ |x\rangle|0\rangle & \text{otherwise} \end{cases}$$

Notice that this is a completely general situation and can be applied not only for the case of finding all the pairs of particles that are close (in which case $|x\rangle = |x_1\rangle|x_2\rangle$, with x_1 and x_2 indices of two particles), but to any setting in which we have to find all the elements in a set that satisfy a certain condition. This is closely related to the Coupon Collector Problem [15], which has been recently studied in a quantum context [16] but with a significant difference: In general, we do not know how many pairs of particles are closer than the threshold, so we are not able to use the methods presented in that work. Another important feature is the fact that, for a given particle, the number of close particles is upper bounded by a constant independent of the total number of particles.

The availability of the oracle O allows us to use Grover's search algorithm [14] that will be central to our methods. It is important to note that the success probability of Grover's algorithm and the number of times it consults the oracle are completely determined by the number of elements ν in the set and by the number μ of *marked* elements (i.e. elements that satisfy the condition). For that reason, in our algorithms, we will consider oracles $O = O_\nu^\mu$ that mark exactly μ elements from a set of size ν. This general setting allows us to consider two different situations: We can search among all the pairs of particles at once (i.e. $\nu = N^2$, and μ is the number of pairs of close particles), or we can fix one of the particles and search for the close ones (i.e. $\nu = N$, and μ is the number of close neighbours). This will prove useful in certain situations, as we explain below, but from the point of view of the analysis of our quantum algorithms, we can consider both cases in just one abstract setting, with the only difference being the values of the parameters ν and μ.

2.1 Oracle construction

In this subsection, we discuss the construction of a quantum circuit implementing the oracle O for the particular case of marking pairs of particles that are below a given distance. In this paper, we will consider that all our algorithms use that circuit as an instantiation of the oracle. Therefore, we want to demonstrate the feasibility of building such an oracle.

A circuit implementing the oracle must return 1 if the distance between two particles i and j is less than or equal to a threshold value δ, and 0 otherwise. That procedure can be divided into two operations: the computation of the distance between i and j and the comparison between that distance and δ. Additionally, as required in two of the proposed algorithms, we will need to modify the oracle O so that once found a marked element x_0, it is excluded from being marked by a new oracle O':

$$O'(|x\rangle|0\rangle) = \begin{cases} |x\rangle|1\rangle & \text{if } x \text{ is marked and } x \neq x_0 \\ |x\rangle|0\rangle & \text{otherwise} \end{cases}$$

Notice that this only implies an extra comparison with x_0, something that can be achieved with a number of gates that is linear on the number of qubits used to represent x_0 and with an additional ancillary qubit.

Focusing on the arithmetic part, the process supports some simplifications. On the one hand, it is possible to work with the squared distances. Therefore, the square root of the distances between particles is not necessary. Then, the distances can be computed using subtractors, adders, and squaring circuits. On the other hand, the comparison can be computed using a comparator circuit. A comparator circuit, also called a full comparator in some sources, receives as input two numbers A and B and determines whether A is less than, equal to, or greater than B [17, 18]. There is a reduced version of this circuit, called half comparator, which only identifies whether A is less than or equal to B, or whether A is greater than B [19]. Half comparators involve fewer resources than full ones [20]. Since in this work it is only necessary to determine

whether the distance is less than or equal to the threshold or not, the comparison can be computed using a half comparator instead of a full comparator.

It is important to note that this oracle will not provide any quantum advantage. However, even quantum circuits that does not provide quantum advantages can be useful as part of larger circuits if they involve a small number of resources [21]. In our case, the oracle must use the least possible number of resources to be efficiently used by our algorithms. In terms of quantum circuits, resource optimization is commonly measured using the number of involved qubits. It is also important to avoid the so-called garbage outputs: qubits that are not part of the result and whose value is not restored to the initial one, so they cannot be used in other parts of the circuit. Reducing the number of operations (represented by the so-called quantum cost) is also desirable [22, 23].

Table 1 shows some of the most prominent adders, subtractors, squaring circuits, and half-comparators available in the literature. The table shows their quantum cost, their number of ancilla inputs, and the number of garbage outputs, according to the definitions given by Mohammadi et al. [22]. Since a complete analysis of the available circuits in the state-of-the-art is out of the scope of this article, we have studied a few selection of them in order to implement a functional oracle. We have followed the methodology described in [24] to measure and to test these circuits. We have chosen the best circuits of each category to build the oracle, prioritizing the absence of garbage outputs and the number of ancilla inputs since their optimization involves fewer qubits.

More specifically, the computation of (squared) Euclidean distances involves subtracting the components of each coordinate, calculating the square of the previous result, and adding these squares. Finally, the obtained value has to be compared with the threshold value. We assume that the squared threshold is entered as an input to the circuit. If this is not the case, an extra circuit will be required to calculate this square. As an example, Fig. 1 shows how the oracle has to be assembled using the above circuits, for the $three$-dimensional case. For a general D-dimensional case, D subtractors will be needed to calculate the difference between each pair of components, D squaring circuits, $D-1$ adders, and a half comparator. To optimize the quantum cost and the number of qubits needed, the circuits to choose are as follows:

- Differences between coordinates: Thapliyal et al. [27](no input carry) ($\overline{a+b}$).
- Squares: Nagamani et al. [34].
- Addition of the squares: Thapliyal et al. [27](no input carry).
- Comparison: Li et al. [19].

Using these circuits, we obtained an oracle with a quantum cost of $D(16N-8)+D(32N)+(D-1)(13N-8)+12N-8$, and with a total of $D+D(6N-3)+(D-1)+1$ ancilla inputs. Finally, it is necessary to uncompute all the qubits (except the one that has the result). The comparator and the squaring circuits include the cost of the uncomputation in their quantum cost and delay, but not the adders. Therefore, an extra quantum cost of $D(16N-8)+(D-1)(13N-8)$ must be considered to avoid any garbage output according to Bennett's garbage removal scheme [38]. No extra qubit is required since one of the ancilla inputs of the comparator is used to keep the result.

Table 1 Evaluation of most optimized circuits, which can be used as part of the oracle O for the general n-digit case, in terms of quantum cost, ancilla inputs and number of garbage outputs

	Circuit	Quantum cost	Ancilla inputs	Garbage outputs
Adders and subtractors	[25] (full subtractors)	$6n$	n	0
	[25] (full and half subtractors)	$6n-2$	n	0
	[26] + [25]	$6n$	$n+1$	0
	[27](input carry) $(\overline{a+b})$	$18n-6$	2	0
	[27](input carry) $(a+\overline{b}+1)$	$16n-4$	2	0
	[27](no input carry) $(\overline{a+b})$	$16n-8$	1	0
	[27](no input carry)	$13n-8$	1	0
	[28] $(\overline{a+b})$	$31n-15W(n)-15log(n)-6$	$5n/4$	0
	[29] $(a+\overline{b}+1)$	$30n-15W(n)-15log(n)-4$	$5n/4$	0
Squaring circuits	[30]	$36n$	$7n$	$7n$
	[31]	$35n$	$10n$	$10n$
	[32]	$36n$	$7n$	$13n$
	[33]	$38n$	$13n$	$13n$
	[34]	$32n$	$6n-3$	0
Half comparators	[35]	$O(n^2)$	$2n$	0
	[36]	$39n+9$	$6n+1$	0
	[23]	$18n+9$	$4n-3$	0
	[37]	$14n$	$4n-2$	0
	[17]	$28n$	2	0
	[27] $(\overline{a+b})$	$32n-18$	3	0
	[27] $(a+\overline{b}+1)$	$30n-10$	3	0
	[25] (full and half subtractors)	$12n$	$2n-3$	0
	[18]	$16n-8$	2	0
	[20]	$13N-12$	N	0
	[19]	$12N-8$	1	0

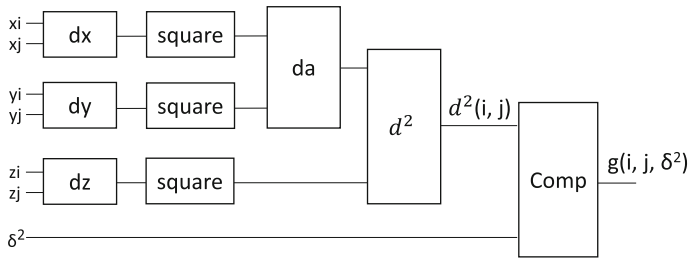

Fig. 1 Scheme of the final circuit for the computation of the oracle, for the $3D$-case

We have built and tested a prototype of the oracle in ProjectQ simulator for a reduced two-dimensional example, $D = 2$. The source code is freely available in https://github.com/2forts/qsec.

2.2 The algorithmic methodology

All our algorithms are based on the use of Grover's search [14]. This quantum algorithm allows, given an oracle O_ν^μ that marks μ elements from a set of size ν, to find, with high probability, a marked element with $O\left(\sqrt{\frac{\nu}{\mu}}\right)$ consults to the oracle, compared to the $\Omega\left(\frac{\nu}{\mu}\right)$ that would be needed with a classical algorithm. This means that there is a quadratic gap between the upper bound of the quantum algorithm, and the lower bound of the classical ones. We will exploit this quadratic speed-up to obtain algorithms that are asymptotically faster than any possible classical algorithm that also uses a black box oracle. Namely, this allows to beat the $\Omega(N^2)$ bound for the search of pairs of close particles in a non-quantum setting. Because of the intrinsic probabilistic nature of quantum computing, our algorithms will provide a correct answer with probability at least $1 - w$, w is a chosen input parameter.

We first consider the situation in which the number of marked elements μ is known. This case will be rarely encountered in practice (when our algorithms are used to find the pairs of particles that are below a given threshold), but we present it here anyway for two reasons. First, it is closely related to the Quantum Coupon Collector Problem that has recently attracted some attention [16]. Second, it will provide a useful benchmark for the more realistic algorithms we present later, as an ideal minimal bound on the number of oracle consults.

Since we are assuming that we know μ, we can simply run Grover's algorithm, checking every time if we have obtained a new marked element, until all of them have been found. However, since Grover's algorithm only returns a marked element with certain probability, there is no upper bound to the number of required oracle consults. For that reason, we propose first to compute a number R of Grover iterations that guarantees to find all marked elements with probability of failure at most w (see the details in "Appendix A"). The complete procedure is, then, the one presented in Algorithm 1.

Algorithm 1: .

Data: An oracle O_ν^μ marking a *known* number of μ elements in a database of ν elements ($0 < \mu \leq \frac{\nu}{2}$). A desired error bound probability $0 < w < 1$.
Result: A set of r marked database elements $L = \{x_1, \ldots, x_r\}$. With probability at least $1 - w$, we will have $r = \mu$.

1 $L \leftarrow \emptyset$;
2 $R \leftarrow \left\lceil \frac{\log\left(\frac{w}{\mu}\right)}{\log\left(1-\frac{1}{2\mu}\right)} \right\rceil$;
3 $done \leftarrow false$;
4 $l \leftarrow 1$;
5 **while** $done = false$ *and* $l \leq R$ **do**
6 Choose j uniformly at random from the set $\{0, \ldots, \lfloor\sqrt{\nu}\rfloor - 1\}$;
7 Run Grover's algorithm with $\left\lceil \frac{\pi}{4}\sqrt{\frac{\nu}{\mu}} \right\rceil$ applications of the oracle plus diffusion operator;
8 Measure to obtain x;
9 **if** x *is a marked element* **then**
10 $L \leftarrow L \cup \{x\}$;
11 **if** $|L| = \mu$ **then**
12 $done \leftarrow true$; /* All marked elements found */
13 **end**
14 **end**
15 $l \leftarrow l + 1$;
16 **end**
17 return L

In practice, however, μ will be unknown to us. This affects our application of Grover's search in two different ways. On the one hand, we can never be sure that we have already found all the marked elements and this affects the stopping conditions (cf. lines 11–13 in Algorithm 1). On the other hand, we do not know what is the optimal number of iterations in Grover's algorithm (cf. line 7 in Algorithm 1). Of course, not knowing μ, also prevents us from computing R.

To overcome these difficulties, we adopt a strategy similar to the one proposed in [39]. For the number of iterations in Grover's search, we select a random number in $\{0, \ldots, \lfloor\sqrt{\nu}\rfloor - 1\}$. For the stopping condition, we compute a value R that will guarantee that if after R executions of Grover's search no marked element has been found, then the probability that indeed there are marked elements is below w, an error bound selected by the user. The mathematical derivation of R is given in "Appendix A". Note that this bound is very conservative and that, in practice, errors much smaller than w will be usually obtained, as shown in the numerical simulations that we have conducted (see Sect. 3). Also note that this extends the method proposed in [39], so that it can be used in our case, we need to find all the elements marked by the oracle (while in [39], only one needs to be found).

The complete procedure is described in Algorithm 2. Notice that in line 20, after a new element has been found, we modify the oracle so that this element is not considered again. For that, we use the construction of the O' oracle mentioned above (Sect. 2.1).

Although Algorithm 2 gives an acceptable worst case asymptotic behaviour (cf. Table 2), the average number of oracle consults can be improved. To this extent, we define a third procedure, Algorithm 3. It uses the techniques proposed in [39] to

Algorithm 2: .

Data: An oracle O_ν^μ marking an *unknown* number of μ elements (upper bounded by a *known or estimated* B) in a database of ν elements ($0 \leq \mu \leq B \leq \frac{3\nu}{4}$). A desired error bound probability $0 < w < 1$.

Result: A set of r marked database elements $L = \{x_1, \ldots, x_r\}$. With probability at least $1 - w$, we will have $r = \mu$.

1 $L \leftarrow \emptyset$;
2 $R \leftarrow \left\lceil \frac{\log\left(1-(1-w)^{\frac{1}{B}}\right)}{\log\left(\frac{3}{4}\right)} \right\rceil$;
3 $found \leftarrow false$;
4 $done \leftarrow false$;
5 **while** $done = false$ **do**
6 $l \leftarrow 1$;
7 **while** $found = false$ *and* $l \leq R$ **do**
8 Choose j uniformly at random from the set $\{0, \ldots, \lfloor\sqrt{\nu}\rfloor - 1\}$;
9 Run Grover's algorithm with j applications of the oracle plus diffusion operator;
10 Measure to obtain x;
11 **if** x *is a marked element* **then**
12 $found \leftarrow true$;
13 **else**
14 $l \leftarrow l + 1$;
15 **end**
16 **end**
17 **if** $found = true$ **then**
18 $L \leftarrow L \cup \{x\}$; /* Add found element and search for another */
19 $found \leftarrow false$;
20 Modify the oracle so that it does not mark x
21 **else**
22 $done \leftarrow true$; /* Tried R times without finding anything */
23 **end**
24 **end**
25 return L

iteratively increase the number of Grover iterations, together with the modification of the oracle and the stopping criterion of checking R additional times that we used in Algorithm 2. Instead of always choosing the number of iterations of Grover's algorithm in a uniform way (see line 8 on Algorithm 2), we now increase the number of iterations, starting from 1, by a factor of $\frac{6}{5}$ (see Algorithm 3, line 27). This allows us to improve the behaviour in the average case, as shown in Table 2. We still need, however, a stopping condition that guarantees that the probability of missing some elements is less than w. This is implemented in the situation when $R > 1$ in the loop of lines 9–18, which is then equivalent to the loop of lines 7–16 in Algorithm 2. Consequently, this leads to a worst case behaviour exactly like that of Algorithm 2. The details of the analysis can be found in "Appendix A".

Table 2 summarizes the oracle query complexities of the three algorithms that we have proposed, where we suppose that, in general, μ is a function of ν.

Algorithm 3: .

Data: An oracle O_ν^μ marking an *unknown* number of μ elements (upper bounded by a *known or estimated* B) in a database of ν elements ($0 \leq \mu \leq B \leq \frac{3\nu}{4}$). A desired error bound probability $0 < w < 1$.

Result: A set of r marked database elements $L = \{x_1, \ldots, x_r\}$. With probability at least $1 - w$, we will have $r = \mu$.

$L \leftarrow \emptyset$;
$m \leftarrow 1$;
$\lambda \leftarrow \frac{6}{5}$;
$R \leftarrow 1$;
$found \leftarrow false$;
$done \leftarrow false$;
while $done = false$ **do**
  $l \leftarrow 1$;
  while $found = false$ *and* $l \leq R$ **do**
    Choose j uniformly at random from the set $\{0, \ldots, \lceil m \rceil - 1\}$;
    Run Grover's algorithm with j applications of the oracle plus diffusion operator;
    Measure to obtain x;
    if *x is a marked element* **then**
      $found \leftarrow true$;
    else
      $l \leftarrow l + 1$;
    end
  end
  if $found = true$ **then**
    $L \leftarrow L \cup \{x\}$; /* Add found element and search for another */
    $m \leftarrow 1$;
    $R \leftarrow 1$;
    $found \leftarrow false$;
    Modify the oracle so that it does not mark x
  else
    if $m < \sqrt{\nu}$ **then**
      $m \leftarrow min(\lambda m, \sqrt{\nu})$; /* Increase Grover iterations limit */
      if $m \geq \sqrt{\nu}$ **then**
        $R \leftarrow \left\lceil \frac{\log\left(1-(1-w)^{\frac{1}{B}}\right)}{\log\left(\frac{3}{4}\right)} \right\rceil$; /* Max Grover iterations reached; try R times */
      end
    else
      $done \leftarrow true$; /* Tried R times without finding anything */
    end
  end
end
return L

2.3 The case of particle pairs

The general search methods presented in the previous subsection can be applied to the problem of determining all the particle pairs that are closer to a given threshold distance. In this paper, the number of close particles to a fixed one is upper bounded by

Table 2 Summary of query complexities (ν is the size of the database, μ is the number of marked elements, $B \leq \frac{3\nu}{4}$ is an upper bound on μ)

Algorithm	Worst case	Average case
1	$O\left(\sqrt{\nu\mu}\log(\mu)\right)$	$O\left(\sqrt{\nu\mu}\log(\mu)\right)$
2	$O\left(\sqrt{\nu}\mu\log(B)\right)$	$O\left(\sqrt{\nu}(\log(B)+\mu)\right)$
3	$O\left(\sqrt{\nu}\mu\log(\nu)\right)$	$O\left(\sqrt{\nu}(\log(\nu)+\sqrt{\mu})\right)$

Table 3 Query complexities in our particular problem, first instantiation: pairs of close particles ($N \geq 54$ is the number of particles, μ is the number of pairs of close particles, $B \leq 27N$ is an upper bound on μ)

Algorithm	Worst case	Average case
1	$O\left(N\sqrt{\mu}\log\mu\right)$	$O\left(N\sqrt{\mu}\log\mu\right)$
2	$O\left(N\mu\log B\right)$	$O\left(N(\log B+\mu)\right.$
3	$O\left(N\mu\log N\right)$	$O\left(N(\log N+\sqrt{\mu})\right)$

Table 4 Query complexities in our particular problem, second instantiation: particles close to a fixed one ($N \geq 54$ is the number of particles, α is the number of particles to search for close neighbours)

Algorithm	Worst case	Average case
1	$O\left(\sqrt{N}\alpha\log\alpha\right)$	$O\left(\sqrt{N}\alpha\log\alpha\right)$
2	$O\left(\sqrt{N}\alpha\log\alpha\right)$	$O\left(\sqrt{N}\alpha\log\alpha\right)$
3	$O\left(\sqrt{N}\log(N)\alpha\log\alpha\right)$	$O\left(\sqrt{N}\log(N)\alpha\log\alpha\right)$

a constant independent of the total number of particles, because of the characteristics of the physical problem (see Sect. 3). We will explore two possible instantiations.

The first one is to consider all possible pairs of particles and apply any of the three algorithms directly. In this case, we will have $\nu = N^2$, where N is the total number of particles, and μ represents the number of pairs of close particles. Provided some mild conditions are met (see "Appendix B"), we obtain the asymptotic complexities shown in Table 3.

In the second instantiation, we fix one particle and search, with any of the three proposed algorithms, for all the particles that are close to it. This can be helpful, as explained in detailed in the next subsection, when only a few of the particles have changed their positions and, thus, we only need to update their neighbour lists. If we consider α to be the number of particles with new positions, then the complexities of the algorithms are those given in Table 4. For the detailed analysis, which is based on the key fact that the number of particles close to a fixed one is upper bounded by a constant independent of the total number of particles, see "Appendix B".

Notice that several of the algorithms offer asymptotic complexities which can be, in the average or even in the worst case, better than those of any classical algorithm (which, necessarily, would have to make $\frac{N(N-1)}{2}$ or αN distance computations and

comparisons). In fact, we will show in Sect. 3 that for a range of parameter values found in real-life problems, our algorithms can greatly reduce the number of oracle queries that need to be performed.

In the next subsection, we explain how the different choices of algorithm can be integrated into a decision procedure depending on the problem parameters and the evolution of the system.

2.4 The decision procedure

As we can see, the second and third algorithms are memory procedures in which the input oracle must be updated in order to keep track of found elements. The three algorithms can be combined with different input parameters in order to obtain the set of close pairs of N particles in the space. Since the particles are continuously moving in space, we propose a two-step dynamic programming strategy: first, looking for close particles among the set of all pairs; later on, looking for close particles to fixed ones, when the positions of particles change (i.e. an update methodology). One aspect to be considered is that Algorithm 3 performs uniformly better than Algorithm 2 in the average case. So, if desired, Algorithm 3 could be a substitute for Algorithm 2 in the alternatives given below.

First step: initialize the pairs of close particles

At this initial stage, the parameter ν is to be instantiated as N^2, and μ is the number of close pairs to be found. The choice of the algorithms is as follows:

- **If** μ is not known, then:
  - **If** μ is believed to be negligible in relation to the total number of pairs, use Algorithm 2 ($O(N)$ oracle calls in the worst case) with an estimated upper bound $B \leq 27N$ of μ.
  - **Else**, use Algorithm 3 with an estimated upper bound $B \leq 27N$ of μ ($O(N\sqrt{N})$ oracle calls in the average case).
- **Else** (μ is known), then:
  - **If** μ is negligible in relation to the total number of pairs, use Algorithm 1 (in the worst scenario, $O(N)$ oracle calls) or Algorithm 2 ($O(N \log N)$ oracle calls in the worst case) with $B = \mu$.
  - **Else**, use Algorithm 1 ($O(N\sqrt{N} \log N)$ oracle calls in the worst case) or Algorithm 3 with $B = \mu$ ($O(N\sqrt{N})$ oracle calls in the average case).

Second step: update the set of particles close to fixed ones

At this stage, the parameter ν is to be instantiated as N, the number of updated particles is α, and for a fixed particle, μ represents the number of close particles to be found.

The alternatives are the following:

1. If $\alpha \log \alpha$ is close to N, then backtrack to the first step.
2. Else, set $S = \left\lceil \frac{\log(\frac{w}{\alpha})}{\log(w)} \right\rceil$. Then:

(a) If μ is known, then use Algorithm 1 S times for each of the α particles ($O(\sqrt{N}\sqrt{\alpha}\log\alpha)$ oracle calls in the worst case).
(b) Else, use Algorithm 2 S times for each of the α particles ($O(\sqrt{N}\sqrt{\alpha}\log\alpha)$ oracle calls in the worst case).

3 Statistical simulation of the algorithms

In this section, the performance of the first-step algorithms introduced in Sect. 2 is tested in practical situations. A key aspect of the simulation is the oracle O, where the particle configuration should be fed into, and the use of Grover's search. For the purpose of testing the actual behaviour of algorithms $1-3$, the oracle is simplified notably, just taking into account the number μ of pairs of close particles, among the total number of N particles. The simulation will simply identify such a number of pairs. Since Grover executions in the algorithms are independent, we can directly simulate (because of the results in [39]) the running of the Grover steps by sampling from a Bernoulli distribution with success probability given by

$$\sin^2((2j+1)\theta)$$

where j is the number of Grover iterations, $\sin^2\theta = \frac{t}{\nu}$ and t is the number of marked elements (notice that $t = \mu$ for Algorithm 1, but in Algorithms 2 and 3 t starts at μ and is decreased by one unit with each element that is found). This means that we do not actually run the Grover steps: we simply simulate the success probability of such runs, instead. In the case of Algorithms 2 and 3 that is enough, because each successful run of Grover will find a different element (we eliminate the obtained ones from the oracle). For Algorithm 1, when the simulation shows that Grover has found a marked element, we sample uniformly from the set $\{1, 2, \ldots, \mu\}$ to determine the actual element that has been found.

In all cases, three values of μ are considered, $\mu = 40$, 80, and 150, while the number of particles N has been chosen to be 125, 216, 512 and 1000. This implies an average number of neighbours per particle ranging from 2.3 to 0.08, which corresponds to some situations found in practice. For instance, in the canonical hard-sphere system, taking a threshold value for the centre-to-centre distance of $3a$, with a the particle radius, these average numbers of neighbours are obtained for volume fractions below 40%.

For Algorithm 1, following the analysis of "Appendix A", the bounds on the total number of iterations for different success probabilities are given in Tables 5, 6 and 7. These bounds, however, are shown to be very conservative once we take into account the actual results found in the simulations. In Tables 8, 9 and 10, we show the minimum, maximum, average and standard deviation of the number of oracle calls needed until all the pairs are found, across 10^6 repetitions of the algorithm. Notice that these values are much lower than those expected from the asymptotic analysis, even when we take into account the standard deviation.

In Table 11, we show the value of R for Algorithms 2 and 3 for $B = 27N$ (recall that R is the number of Grover iterations that we need to execute without finding

Table 5 Bounds on # of oracle calls for Algorithm 1 when $\mu = 40$

Error bound w	# Calls 125 part	# Calls 216 part	# Calls 512 part	# Calls 1000 part
0.1	7632	15,264	30,528	61,056
0.05	8512	17,024	34,048	68,096
0.01	10,560	21,120	42,240	84,480
0.005	11,440	22,880	45,760	91,520
0.001	13,488	26,976	53,952	107,094

Table 6 Bounds on # of oracle calls for Algorithm 1 when $\mu = 80$

Error bound w	# Calls 125 part	# Calls 216 part	# Calls 512 part	# Calls 1000 part
0.1	12,804	24,541	48,015	96,030
0.05	14,124	27,071	52,965	105,930
0.01	17,208	32,982	64,530	129,060
0.005	18,540	35,535	69,525	139,050
0.001	21,612	41,423	81,045	162,090

Table 7 Bounds on # of oracle calls for Algorithm 1 when $\mu = 150$

Error bound w	# Calls 125 part	# Calls 216 part	# Calls 512 part	# Calls 1000 part
0.1	19,719	37,247	72,303	144,606
0.05	21,582	40,766	79,134	158,268
0.01	25,920	48,960	95,040	190,080
0.005	27,792	52,496	101,904	203,808
0.001	32,130	60,690	117,810	235,620

Table 8 Minimum, maximum, average and standard deviation of the number of iterations for 10^6 repetitions of Algorithm 1 when $\mu = 40$

Particles	Minimum	Maximum	Average	Standard deviation
125	928	12,600	2749.08	790.33
216	1888	24,224	5481.58	1575.03
512	3904	44,928	10957.61	3150.78
1000	7552	86,144	21909.18	6313.69

any new particle in order to stop, and that B is the bound on the number of particles that are close to each other). Again, these bounds prove to be extremely conservative. We have executed Algorithms 2 and 3 for 10^6 times with values of R taken from $\{5, 10, \ldots, 70\}$. The full results can be found in the supplementary material. In this section, we present only the data for the first value of R that successfully finds all the particle pairs in all 10^6 experiments for a fixed value of μ. Since all these results can be quickly obtained from simulations alone, for other values of N, ν and μ, one can

Table 9 Minimum, maximum, average and standard deviation of the number of iterations for 10^6 repetitions of Algorithm 1 when $\mu = 80$

Particles	Minimum	Maximum	Average	Standard deviation
125	1908	20,064	4920.50	1243.43
216	3795	33,833	9181.87	2318.84
512	7254	61,650	17,887.36	4516.25
1000	14,940	131,490	35,743.77	9016.89

Table 10 Minimum, maximum, average and standard deviation of the number of iterations for 10^6 repetitions of Algorithm 1 when $\mu = 150$

Particles	Mininum	Maximum	Average	Standard deviation
125	3636	28,665	8038.76	1819.60
216	6613	49,691	14415.13	3266.95
512	12,606	92,532	27695.03	6265.49
1000	24,354	180774	55391.35	12542.27

Table 11 Number of repetitions for different error bounds in Algorithms 2 and 3 when $\mu = 40$

Error bound w	R 125 part	R 216 part	R 512 part	R 1000 part
0.1	37	39	41	44
0.05	39	42	44	46
0.01	45	47	50	52
0.005	47	50	52	54
0.001	53	55	58	60

repeat experiments similar to the ones presented here in order to determine, before using an actual quantum computer, which algorithm is most suitable for the situation and what is the desirable value of R. In Figs. 2, 3 and 4 , we show those results, including the value of R and the minimum, maximum, average and standard deviation of the number of oracle calls used by the algorithms.

We can see that as it was the case with Algorithm 1, Algorithms 2 and 3, we achieve an error rate below one in a million for values of R much less than what Table 11 would lead to expect.

In Figs. 5, 6 and 7, we compare the number of queries needed by the classical algorithm with the average number of queries made by Algorithms 1, 2 and 3. Notice that while the growth in the case of the classical algorithm is quadratic, for our algorithms, it is linear for fixed values of μ. In fact, for the lowest values of μ, the average number of queries of all our algorithms is lower than the number of queries performed by the classical algorithm. For bigger values of μ (80 and 150), the classical algorithm beats some of the quantum algorithms for low number of particles (125 and 216) but for the simulations with 512 and 1000 particles, our algorithms are always better (and

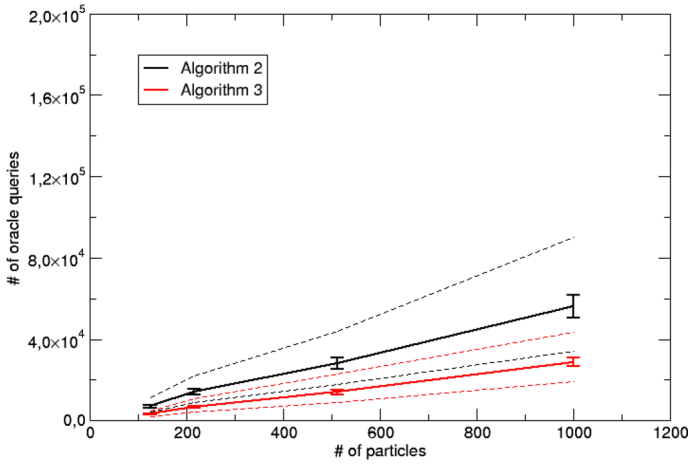

Fig. 2 Number of oracle queries of Algorithms 2 (black) and 3 (red) for different numbers N of particles when $\mu = 40$. The solid line is the average, the dashed lines are the minimum and maximum, and the bars represent the standard deviation (Color figure online)

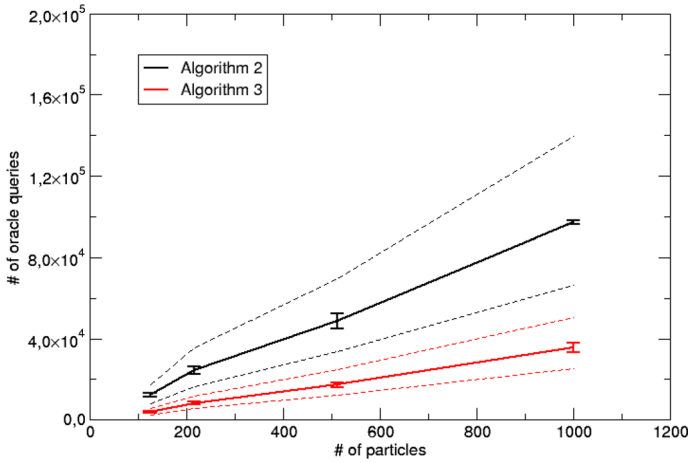

Fig. 3 Number of oracle queries of Algorithms 2 (black) and 3 (red) for different numbers N of particles when $\mu = 80$. The solid line is the average, the dashed lines are the minimum and maximum, and the bars represent the standard deviation (Color figure online)

the speed-up increases with the number of particles). In fact, Algorithm 3 was always better than the classical algorithm for all the cases under study.

These results clearly show that our algorithms can outperform the best classical algorithm in terms of oracle queries when the density of particles is low (μ is low or ν is high). Thus, once robust quantum hardware is available, these methods, especially Algorithm 3, may be of use in practical situations, where the density is usually low, a situation in which our algorithms show their better performance.

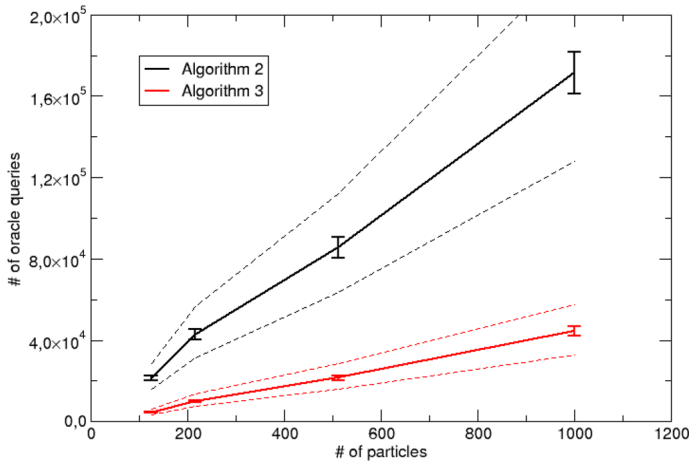

Fig. 4 Number of oracle queries of Algorithms 2 (black) and 3 (red) for different numbers N of particles when $\mu = 150$. The solid line is the average, the dashed lines are the minimum and maximum, and the bars represent the standard deviation (Color figure online)

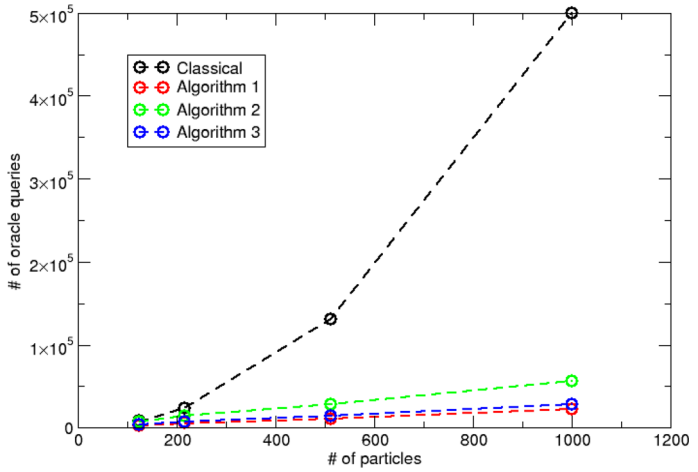

Fig. 5 Comparison of the number of oracle queries of the different algorithms when $\mu = 40$

4 Conclusions

The focus of this work has been on the use of quantum computing to efficiently calculate the neighbour list in the context of N-body simulations. A quantum algorithm based on oracle procedures (Grover) has been considered to carry out the whole proposal. The oracle has been designed with efficient reversible circuits that identify if pairs of bodies are neighbours or not. A prototype of the oracle has been developed in ProjectQ simulator based on the circuits proposed in [18, 27, 34] and it is available at https://github.com/2forts/qsec. Three quantum algorithms have been designed to get

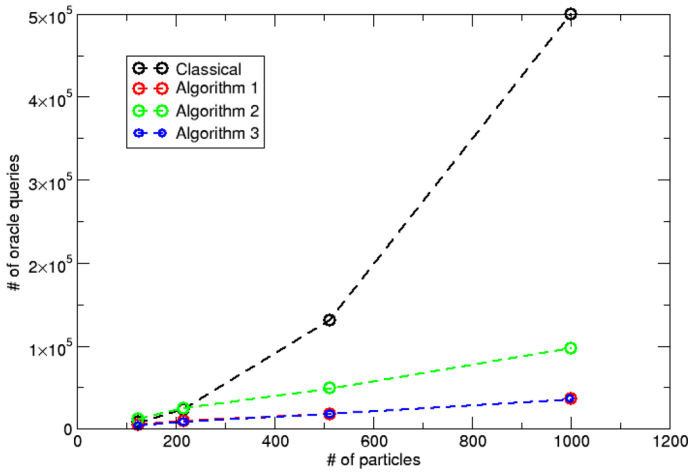

Fig. 6 Comparison of the number of oracle queries of the different algorithms when $\mu = 80$

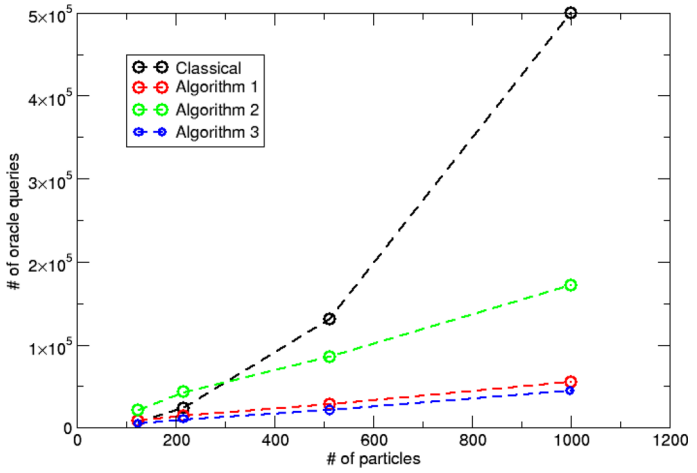

Fig. 7 Comparison of the number of oracle queries of the different algorithms when $\mu = 150$

the pairs of neighbour particles from the information provided by the oracle. They can be combined in a two-step procedure for achieving such an objective: first, looking for pairs of close particles; second, updating the neighbour list of a small number of particles that move beyond a certain threshold. The actual combination of the algorithms has been described in a decision procedure that aims to provide the best algorithm for each possible situation.

The asymptotic analysis of every algorithm has been justified from a theoretical point of view. A statistical simulation of the oracle O in combination with the algorithms has been considered to test their statistical behaviour for μ pairs of close particles, among N particles.

After 10^6 repetitions of the algorithms, we have computed the minimum, maximum, average and standard deviation of the number of oracle calls needed until all the pairs were found. The obtained values have been much lower than those expected from the asymptotic analysis.

Thus, once robust quantum hardware is available, these methods, especially Algorithm 3, may be of use in practical situations, where the density is usually low, a situation in which our algorithms have shown their best performance.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s11128-023-04245-1.

Acknowledgements This work has been partially supported by the Spanish Ministry of Science and Innovation throughout Project PID2021-123278OB-I00 (funded by MCIN/AEI/10.13039/501100011033/ FEDER "A way to make Europe"), PID2020-119082RB-C22 (funded by MCIN/AEI/10.13039/501100011033), Projects PID2021-123461NB-C22, and PID2021-127836NB-I00; by the Regional Ministry of Junta de Andalucía under the Grants PY20_00748, IC-DRUGS-P18-RT-1193, UAL2020-TIC-A2101, and UAL18-TIC-A020-B; by the Regional Ministry of the Principado de Asturias under Grant AYUD/2021/50994, by the European Regional Development Fund (ERDF), and by the QUANTUM SPAIN project funded by the Ministry of Economic Affairs and Digital Transformation of the Spanish Government and the European Union through the Recovery, Transformation and Resilience Plan.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Data availability Data sharing is not applicable to this article as no datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

Appendix A. Mathematical proof of the asymptotic behaviour of the proposed quantum algorithms

Algorithm 1

Given a database of ν unsorted elements and an oracle that detects $\mu = \mu(\nu)$ marked elements, Algorithm 1 provides a method that finds all marked elements with a bounded probability error, based on the repeated use of Grover's algorithm. We shall require that for all ν, $0 < \mu(\nu)$. We will also assume that the sequence $\mu(\nu)$ has a limit when $\nu \to \infty$.

Grover's algorithm provides, with $O\left(\sqrt{\frac{\nu}{\mu(\nu)}}\right)$ oracle calls, a success probability greater or equal than $\delta(\nu) := 1 - \frac{\mu(\nu)}{\nu}$, i.e. $\delta(\nu) := P$(finding a marked element out

Table 12 Summary of Algorithm 1

# Marked elements	#Iterations	#Orac. calls per it	Total # oracle calls
$\mu(\nu)$	$O\left(\mu(\nu)\log(\mu(\nu))\right)$	$O\left(\sqrt{\frac{\nu}{\mu(\nu)}}\right)$	$O\left(\sqrt{\nu\mu(\nu)}\log(\mu(\nu))\right)$

of the $\mu(\nu)$) [39, Section 3]. Assuming that $\mu(\nu) \leq \frac{\nu}{2}$, for all ν, we have a uniformly bounded success probability $\delta(\nu) \geq \frac{1}{2}$. Because such an algorithm does not distinguish between marked elements, we have that

$$P_i(\nu) := P(\text{finding the } i-\text{th marked element out of the } \mu(\nu)) = \frac{\delta(\nu)}{\mu(\nu)} \geq \frac{1}{2\mu(\nu)}$$

for all $i = 1, \ldots, \mu(\nu)$, and for all ν. We want to independently repeat the search $R = R(\nu)$ times and estimate the probability $P'(\nu)$ of not finding all marked elements. Namely,

$$\begin{aligned} P'(\nu) &:= P(\text{not finding all marked elements in } R(\nu) \text{ experiments}) \\ &= P(\text{not find. the first elem. in } R(\nu) \text{ exp. } \vee \ \ldots \ \vee \text{ not find. the} \\ \mu(\nu) - \text{th elem. in } R(\nu) \text{ exp.}) &\leq \mu(\nu)\left(1 - \frac{1}{2\mu(\nu)}\right)^{R(\nu)} \end{aligned}$$

In order to obtain a bounded algorithm, we require that such a probability is less than some $w < 1$, for all ν. This yields $\mu(\nu)\left(1 - \frac{1}{2\mu(\nu)}\right)^{R(\nu)} \leq w$ or, equivalently,

$$R(\nu) \geq \frac{\log\left(\frac{w}{\mu(\nu)}\right)}{\log\left(1 - \frac{1}{2\mu(\nu)}\right)}$$

Taking $R(\nu)$ as $\left\lceil \frac{\log\left(\frac{w}{\mu(\nu)}\right)}{\log\left(1-\frac{1}{2\mu(\nu)}\right)} \right\rceil$, we have that $R(\nu) = O\left(\mu(\nu)\log(\mu(\nu))\right)$, and the procedure requires an overall number of $O\left(\sqrt{\nu\mu(\nu)}\log(\mu(\nu))\right)$ oracle calls (Table 12).

The main obstacles to a practical application of this methodology are the requirements on $\mu(\nu)$, namely it has to be *known* and satisfy $0 < \mu(\nu) \leq \frac{\nu}{2}$, for all ν. Moreover, the correctness of the asymptotic analysis is conditioned to the sequence $\mu(\nu)$ having a limit. Since $\mu(\nu)$ is not always known, Algorithms 2 and 3 give two practical approaches based on Grover's algorithm with a random number of iterations. In both cases, an algorithm with memory and an appropriate time-out is adopted.

Algorithm 2

This algorithm consists in a direct randomization of the number of Grover's iterations of Algorithm 1. The list L keeps track of marked elements already found (a memory list), and the number $R = R(\nu)$ of times that Grover's search is repeated has to be defined so that the algorithm has a bounded success probability. This time we shall

Table 13 Summary of Algorithm 2: worst case

#Iterations loop lines 5–24	#Iterations loop lines 7–16	#Orac. calls per it	Total # oracle class
$\mu(\nu)+1$ (output iter.)	$O\left(\log(B(\nu))\right)$	$O\left(\sqrt{\nu}\right)$	$O\left(\sqrt{\nu}\mu(\nu)\log(B(\nu))\right)$

require that, for all ν, $0 < \mu(\nu) \leq \frac{3\nu}{4}$, and that the sequence $\mu(\nu)$ has a limit when $\nu \to \infty$.

Let us consider the correctness of the algorithm. At any given moment, the number of elements marked by the oracle is $0 \leq t \leq \frac{3\nu}{4}$. When $t = 0$, the loop on lines 7–15 will not find any marked element. Thus, the conditional on line 17 will take the "else" branch and the algorithm will output L on line 25. Since $t = 0$, L will contain all the originally marked elements. On the other hand, when $t > 0$, because of Lemma 2 and the proof of Theorem 3 in [39], the probability of finding a marked element is $\delta(\nu) \geq \frac{1}{4}$, with $O\left(\sqrt{\nu}\right)$ oracle calls, so the overall probability of finding a marked element is $1-(1-\delta(\nu))^{R(\nu)} \geq 1-\left(\frac{3}{4}\right)^{R(\nu)}$.

Since the loop on lines 5–24 must be independently repeated $\mu(\nu)+1$ times for the algorithm to succeed (the last iteration is the one forcing the output), the probability $P'(\nu)$ of not finding all marked elements is $P'(\nu) := 1-\left(1-(1-\delta(\nu))^{R(\nu)}\right)^{\mu(\nu)} \leq 1-\left(1-\left(\frac{3}{4}\right)^{R(\nu)}\right)^{\mu(\nu)}$ which, in order to obtain a bounded algorithm, is required to be less than some $w < 1$, for all ν. This yields

$$R(\nu) \geq \frac{\log\left(1-(1-w)^{\frac{1}{\mu(\nu)}}\right)}{\log\left(\frac{3}{4}\right)}$$

Taking $R(\nu)$ to be $\left\lceil \frac{\log\left(1-(1-w)^{\frac{1}{\mu(\nu)}}\right)}{\log\left(\frac{3}{4}\right)} \right\rceil$, we have that $R(\nu) = O\left(\log(\mu(\nu))\right)$, and the procedure requires an overall number of $O\left(\sqrt{\nu}\mu(\nu)\log(\mu(\nu))\right)$ oracle calls. Of course, since $\mu(\nu)$ is assumed to be unknown, in practice we should use an upper bound $B(\nu)$ of $\mu(\nu)$. (In the worst case, we can always choose $B(\nu) = \frac{3\nu}{4}$.) This allows us to take $R(\nu) = \left\lceil \frac{\log\left(1-(1-w)^{\frac{1}{B(\nu)}}\right)}{\log\left(\frac{3}{4}\right)} \right\rceil = O(\log(B(\nu))$, and the overall asymptotic complexity is $O\left(\sqrt{\nu}\mu(\nu)\log(B(\nu))\right)$ (Table 13).

In this algorithm, it is also interesting to analyse the average number of oracle queries. Since the probability of finding an element in any of the Grover executions of lines 9–10 is at least $\frac{1}{4}$, the average number of queries on each execution of the loop of lines 7–16 is less than $4\frac{\sqrt{\nu}}{2} = 2\sqrt{\nu}$ when there are still marked elements to be found. We need to add to that the number of queries of the output iteration (when all elements have already been found) to obtain an average number of queries which is $2\sqrt{\nu}\mu(\nu) + O\left(\sqrt{\nu}\log(B(\nu))\right) = O\left(\sqrt{\nu}(\log(B(\nu)) + \mu(\nu))\right)$ (Table 14).

Table 14 Summary of Algorithm 2: average case

#Iterations loop lines 5–24	#Iterations loop lines 7–16	#Orac. calls per it	Total # oracle class
$\mu(\nu)+1$ (output iter.)	4 or $O(\log(B(\nu)))$	$\frac{\sqrt{\nu}}{2}$ or $\sqrt{\nu}$	$O(\sqrt{\nu}(\log(B(\nu))+\mu(\nu)))$

Table 15 Summary of Algorithm 3: worst case (noncritical and critical stages)

#Iterations loop lines 7–35	#Iter. loop 9–18 to reach the critical stage	#Orac. calls per it	Total # oracle calls
$\mu(\nu)+1$ (output iter.)	$O(\log(\nu))$	$O\left(\sqrt{\nu}\right)$	$O\left(\sqrt{\nu}\mu(\nu)\log(\nu)\right)$
#Iterations loop lines 7–35	#Iterations loop lines 9–18	#Orac. calls per it	Total # oracle calls
$\mu(\nu)+1$ (output iter.)	$O(\log(B(\nu)))$	$O\left(\sqrt{\nu}\right)$	$O\left(\sqrt{\nu}\mu(\nu)\log(B(\nu))\right)$

The main obstacles to a practical application of this methodology are the requirements on $\mu(\nu)$, as it has to satisfy $0 < \mu(\nu) \leq \frac{3\nu}{4}$, for all ν; the asymptotic behaviour of the algorithm, which is worst than in the straightforward approach; the need of a continuous oracle update. The main advantages are that $\mu(\nu)$ is now not required to be known, and that the sequence $\mu(\nu)$ is not required to have a limit, when $\nu \to \infty$.

Algorithm 3

This alternate algorithm is a variation of the previous one, based on [39], and it consists in two stages. In the first one, the parameter m increases from 1 to $\sqrt{\nu}$ by a factor of λ. In each iteration, Grover's algorithm is only run once. When the *critical* stage is reached (i.e. when $m = \sqrt{\nu}$), the algorithm behaves exactly as the previous one. Since the algorithm never outputs before reaching the critical stage, the error probability is bounded as above. The difference here consists on the number of oracle calls. In the worst case, the algorithm performs the number of calls of the previous algorithm plus the oracle calls of the noncritical stage, but this latter number is $O\left(\sqrt{\nu}\log(\nu)\right)$, since $O(\log(\nu))$ iterations are needed to reach the critical stage. So the overall complexity of the worst case is $O\left(\sqrt{\nu}\mu(\nu)\log(\nu)\right)$ (Table 15).

Again, the average number of queries can be substantially lower than that. Indeed, from Theorem 3 in [39], when there are $t > 0$ marked elements to be found, the average number of oracle queries that our algorithm needs to perform in order to find one of them is $O\left(\sqrt{\frac{\nu}{t}}\right)$. Hence, the average number of queries is $O\left(\sum_{t=1}^{\mu(\nu)}\sqrt{\frac{\nu}{t}}\right) + O\left(\sqrt{\nu}\log(\nu)\right) + O\left(\sqrt{\nu}\log(B(\nu))\right) = O\left(\sqrt{\nu\mu(\nu)}\right) + O\left(\sqrt{\nu}\log(\nu)\right) = O\left(\sqrt{\nu}(\log(\nu)+\sqrt{\mu(\nu)})\right)$, because $B(\nu) = O(\nu)$ (see Table 16).

The obstacles to a practical application of this algorithm are mostly the ones of the previous one. However, although its asymptotic number of calls is never smaller than the algorithm above, its average number of queries can be better in practice (this has been observed in simulations). In fact, even though the worst case query complexity

Table 16 Summary of Algorithm 3: average case

#Iterations loop 9–18	#Orac. calls per it	Total # oracle class
$t = 1, \ldots, \mu(\nu)$	$\sqrt{\frac{\nu}{t}}$	$O\left(\sqrt{\nu\mu(\nu)}\right)$
1 (output iter.)	$\sqrt{\nu}\log(\nu)$ (noncritical) + $\sqrt{\nu}\log(B(\nu))$	$O(\sqrt{\nu}\log(\nu))$

Table 17 Summary of query complexities ($B(\nu) \leq \frac{3\nu}{4}$ is an upper bound of $\mu(\nu)$)

Algorithm	Worst case	Average case
1	$O\left(\sqrt{\nu\mu(\nu)}\log(\mu(\nu))\right)$	$O\left(\sqrt{\nu\mu(\nu)}\log(\mu(\nu))\right)$
2	$O\left(\sqrt{\nu}\mu(\nu)\log(B(\nu))\right)$	$O\left(\sqrt{\nu}(\log(B(\nu)) + \mu(\nu))\right)$
3	$O\left(\sqrt{\nu}\mu(\nu)\log(\nu)\right)$	$O\left(\sqrt{\nu}(\log(\nu) + \sqrt{\mu(\nu)})\right)$

is worse than that of the first algorithm proposed, the average number of queries is better when $\log(\nu) + \sqrt{\mu(\nu)}$ is $o(\sqrt{\mu(\nu)}\log(\mu(\nu)))$.

Summary of complexities
In Table 17, we provide a table that summarizes the complexities of the three algorithms that we have proposed.

Appendix B. Rationale behind the decision procedure

As mentioned in the text, the decision procedure for the determination of pairs of close particles consists in two steps. First, look for close particles among the set of all pairs. Second, look for close particles to a fixed one, when the positions of particles change (i.e. an update methodology). In each case, any of the three methods above can be potentially used. Next we explain the rationale behind our proposal.

First step: look directly for pairs of close particles
In this case, $\nu = N^2$, and the required bounds on $\mu(N^2)$ are always satisfied when the number of particles is $N \geq 54$ (for the first algorithm) or $N \geq 36$ (for the second and third ones), because of the characteristics of the physical problem (see Sect. 3). However, for smaller sizes of the problem and particularly small values of $\mu(N^2)$, the algorithms could still work. The assumption that $\mu(N^2)$ has a limit, as $N^2 \to \infty$, is realistic since the density is fixed, namely the ratio of number of particles to available space is constant. Therefore, the more particles we have, the more chances of having pairs of close particles, i.e. it seems realistic assuming that $\mu(N^2)$ is non-decreasing, and so it has a limit. The main obstacle for using the first algorithm is the need of a knowledge of the actual value of $\mu(N^2)$. The asymptotic number of oracle calls of each algorithm is given in Table 18

Depending on the actual $\mu(N^2)$, we will have different complexities. For instance, it has been noticed in practice that sometimes the number of close pairs of *distinct* particles is small in relation to the total number of pairs. This can be translated as the condition $\mu(N^2) = O(1)$ (since we do not count the N pairs of a repeated particle),

Table 18 Query complexities in our particular problem

Algorithm	Worst case	Average case
1	$O\left(N\sqrt{\mu(N^2)}\log(\mu(N^2))\right)$	$O\left(N\sqrt{\mu(N^2)}\log(\mu(N^2))\right)$
2	$O\left(N\mu(N^2)\log(B(N^2))\right)$	$O\left(N(\log(B(N^2))+\mu(N^2)\right)$
3	$O\left(N\mu(N^2)\log(N)\right)$	$O\left(N(\log(N)+\sqrt{\mu(N^2)})\right)$

Table 19 Query complexities when $\mu(N^2)$, $B(N^2) = O(1)$, or $\mu(N^2)$, $B(N^2) = O(N)$

Algorithm	Worst case	Average case	$\mu(N^2)$, $B(N^2)$
1	$O(N)$	$O(N)$	$O(1)$
2	$O(N)$	$O(N)$	
3	$O(N\log(N))$	$O(N\log(N))$	
1	$O\left(N\sqrt{N}\log(N))\right)$	$O\left(N\sqrt{N}\log(N))\right)$	$O(N)$
2	$O\left(N^2\log(N)\right)$	$O\left(N^2\right)$	
3	$O\left(N^2\log(N)\right)$	$O\left(N\sqrt{N}\right)$	

and so the number of oracle calls, in both the worst and average cases, is simply $O(N)$ for the first two algorithms (observe that $\mu(N^2) = O(1)$ allows $B(N^2)$ to be taken as $O(1)$) and $O(N\log(N))$ for the third one. In this situation, it seems reasonable to expect that the three algorithms might give accurate outputs even for small values of N.

On the other hand, we might simply assume that $\mu(N^2) = O(N)$ (because of the uniform bound on the number of particles close to a fixed one), and so the algorithms require queries of the orders given in Table 19. Notice that, in this case, algorithm 2 (taking the natural choice $B(N^2) = O(N)$) should be avoided, and one can choose between algorithm 1 (in a conservative setting, and if the exact value of $\mu(N^2)$ is known) and algorithm 3 (if only the average running time is of interest).

Second step: fix one particle and look for the close ones

Here, we have $\nu = N$ and $\mu(N) \leq 27$. If we want to apply the general setting, the requirement on the minimum number of particles is the same as above ($N \geq 54$ for the first algorithm and $N \geq 36$ for the second and third ones). Also, for the first method, we need to assume that $\mu(N)$ has a limit, as $N \to \infty$. Again, this assumption is realistic since the more particles we have, the more chances of having close particles to a given one, i.e. it seems realistic assuming that $\mu(N)$ is non-decreasing, and so it has a limit. Moreover, in this situation, $\mu(N) = O(1)$ always. The need of a knowledge of $\mu(N)$ is, as above, the main obstacle for using the first algorithm.

Application of the general setting yields an asymptotic number of oracle calls that is $O\left(\sqrt{N}\right)$ for the first two methods, and $O(\sqrt{N}\log(N))$ for the third one. This number of oracle queries has to be multiplied by the number of "updated" particles

that we will call $\alpha(N)$. There is still another missing factor that must be taken into account. We know that any of the algorithms provides a uniform success probability $0 < 1 - w < 1$. When we repeat the algorithm $\alpha(N)$ times, the lower bound on the success probability becomes $(1-w)^{\alpha(N)}$, which tends to 0, as $\alpha(N)$ tends to infinity. To avoid this, we can repeat the search method S times for each updated particle, so that the probability that we do not find all the close pairs is bounded from above by $\sum_{i=1}^{\alpha(N)} P(\text{fail to find the neighbour list of the i-th particle in all the S repetitions}) = \alpha(N)w^S$. Then, if we take $S = \left\lceil \frac{\log(\frac{\epsilon}{\alpha(N)})}{\log(w)} \right\rceil$, which is $O(\log(\alpha(N)))$, we can make the failure probability less than any given ϵ, in particular w. Therefore, the total amount of oracle calls that we need to consider is $O\left(\sqrt{N}\alpha(N)\log(\alpha(N))\right)$ for the first two algorithms and $O\left(\sqrt{N}\log(N)\alpha(N)\log(\alpha(N))\right)$ for the third one.

Backtracking

A final question to be addressed is when it would be desirable to retake the first approach instead of updating with the second approach. This would happen, for instance, when the number of updated particles, $\alpha(N)$, verifies $\alpha(N)\log(\alpha(N)) \geq N$, but the constants hidden by the O notation can make it interesting even for smaller $\alpha(N)$.

References

1. March, N., Tosi, M.: Atomic Dynamics in Liquids. Dover Publications, New York (1991)
2. Allen, M., Tildesley, D.: Computer Simulation of Liquids. Clarendon Press, Oxford (1989)
3. Hayes, B.: The 100-billion-body problem. Am. Sci. **103**(90)
4. Caballero, J.B., Puertas, A.M., Fernández-Barbero, A., Javier de las Nieves, F.: Formation of clusters in a mixture of spherical colloidal particles oppositely charged. Colloids Surfaces A Physicochem. Eng. Asp. **270–271**, 285–290 (2005)
5. Barnes, J., Hut, P.: A hierarchical o(n log n) force-calculation algorithm. Nature **324**, 446–449 (1986)
6. Dehnen, W.: A hierarchical o(n) force calculation algorithm. J. Comput. Phys. **179**(1), 27–42 (2002)
7. Chialvo, A.A., Debenedetti, P.G.: On the use of the Verlet neighbor list in molecular dynamics. Comput. Phys. Commun. **60**(2), 215–224 (1990). https://doi.org/10.1016/0010-4655(90)90007-N
8. Howard, M.P., Anderson, J.A., Nikoubashman, A., Glotzer, S.C., Panagiotopoulos, A.Z.: Efficient neighbor list calculation for molecular simulation of colloidal systems using graphics processing units. Comput. Phys. Commun. **203**, 45–52 (2016)
9. Potestio, R., Peter, C., Kremer, K.: Computer simulations of soft matter: linking the scales. Entropy **16**, 4199–4245 (2014). https://doi.org/10.3390/e16084199
10. Nielsen, M.A., Chuang, I.: Quantum computation and quantum information (2002)
11. Paredes, B., Verstraete, F., Cirac, J.I.: Exploiting quantum parallelism to simulate quantum random many-body systems. Phys. Rev. Lett. **95**(14), 140501 (2005)
12. Lidar, D.A., Rezakhani, A.T., Hamma, A.: Adiabatic approximation with exponential accuracy for many-body systems and quantum computation. J. Math. Phys. **50**(10), 102106 (2009)
13. Sørensen, A.S., Altman, E., Gullans, M., Porto, J., Lukin, M.D., Demler, E.: Adiabatic preparation of many-body states in optical lattices. Phys. Rev. A **81**(6), 061603 (2010)
14. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pp. 212–219 (1996)
15. Isaac, R.: The pleasures of probability, Undergraduate Texts in Mathematics. Springer, New York (1995), readings in Mathematics. https://doi.org/10.1007/978-1-4612-0819-8
16. Arunachalam, S., Belovs, A., Childs, A.M., Kothari, R., Rosmanis, A., de Wolf, R.: Quantum coupon collector. In: Flammia, S.T. (ed.) 15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020), Vol. 158 of Leibniz International Proceedings in Informatics

(LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 10:1–10:17 (2020). https://doi.org/10.4230/LIPIcs.TQC.2020.10. https://drops.dagstuhl.de/opus/volltexte/2020/12069
17. Xia, H., Li, H., Zhang, H., Liang, Y., Xin, J.: An efficient design of reversible multi-bit quantum comparator via only a single ancillary bit. Int. J. Theor. Phys. **57**(12), 3727–3744 (2018)
18. Xia, H., Li, H., Zhang, H., Liang, Y., Xin, J.: Novel multi-bit quantum comparators and their application in image binarization. Quantum Inf. Process. **18**(7), 229 (2019)
19. Li, H.-S., Fan, P., Xia, H., Peng, H., Long, G.-L.: Efficient quantum arithmetic operation circuits for quantum image processing. SCIENCE CHINA Phys. Mech. Astron. **63**, 1–13 (2020)
20. Orts, F., Ortega, G., Cucura, A., Filatovas, E., Garzón, E.: Optimal fault-tolerant quantum comparators for image binarization. J. Supercomput. 1–12 (2021)
21. Pérez-Salinas, A., Cervera-Lierta, A., Gil-Fuster, E., Latorre, J.I.: Data re-uploading for a universal quantum classifier. Quantum **4**, 226 (2020)
22. Mohammadi, M., Eshghi, M.: On figures of merit in reversible and quantum logic designs. Quantum Inf. Process. **8**(4), 297–318 (2009)
23. Thapliyal, H., Ranganathan, N., Ferreira, R.: Design of a comparator tree based on reversible logic. In: 10th IEEE International Conference on Nanotechnology, pp. 1113–1116. IEEE (2010)
24. Orts, F., Ortega, G., Combarro, E.F., Garzón, E.M.: A review on reversible quantum adders. J. Netw. Comput. Appl. **170**, 102810 (2020). https://doi.org/10.1016/j.jnca.2020.102810
25. Thapliyal, H.: Mapping of subtractor and adder-subtractor circuits on reversible quantum gates. In: Transactions on Computational Science XXVII, pp. 10–34. Springer, New York (2016)
26. Orts, F., Ortega, G., Garzón, E.M.: A faster half subtractor circuit using reversible quantum gates. Baltic J. Mod. Comput. **7**(1), 99–111 (2019)
27. Thapliyal, H., Ranganathan, N.: Design of efficient reversible logic-based binary and BCD adder circuits. ACM J. Emerg. Technol. Comput. Syst. (JETC) **9**(3), 17 (2013)
28. Thapliyal, H., Jayashree, H., Nagamani, A., Arabnia, H.R.: Progress in reversible processor design: a novel methodology for reversible carry look-ahead adder. In: Transactions on Computational Science XVII, pp. 73–97. Springer, New York (2013)
29. Draper, T.G., Kutin, S.A., Rains, E.M., Svore, K.M.: A logarithmic-depth quantum carry-lookahead adder. arXiv preprint arXiv:quant-ph/0406142
30. Bhagyalakshmi, H., Venkatesha, M.: Optimized multiplier using reversible multi-control input Toffoli gates. Int. J. VLSI Des. Commun. Syst. **3**(6), 27 (2012)
31. Rangaraju, H., Suresh, A.B., Muralidhara, K.: Design and optimization of reversible multiplier circuit. Int. J. Comput. Appl. **52**(10), 44–50 (2012)
32. Islam, M.S., Rahman, M., Begum, Z., Hafiz, M.Z.: Low cost quantum realization of reversible multiplier circuit. Inf. Technol. J. **8**(2), 208–213 (2009)
33. Bhagyalakshmi, H., Venkatesha, M.: An improved design of a multiplier using reversible logic gates. Int. J. Eng. Sci. Technol. **2**(8), 3838–3845 (2010)
34. Nagamani, A., Ramesh, C., Agrawal, V.K.: Design of optimized reversible squaring and sum-of-squares units. Circuits Syst. Signal Process. **37**(4), 1753–1776 (2018)
35. Wang, D., Liu, Z.-H., Zhu, W.-N., Li, S.-Z.: Design of quantum comparator based on extended general Toffoli gates with multiple targets. Comput. Sci. **39**(9), 302–306 (2012)
36. Al-Rabadi, A.N.: Closed-system quantum logic network implementation of the Viterbi algorithm. Facta Universitatis-Series Electron. Energet. **22**(1), 1–33 (2009)
37. Vudadha, C., Phaneendra, P.S., Sreehari, V., Ahmed, S.E., Muthukrishnan, N.M., Srinivas, M.B.: Design of prefix-based optimal reversible comparator. In: IEEE Computer Society Annual Symposium on VLSI, pp. 201–206. IEEE (2012)
38. Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev. **17**(6), 525–532 (1973)
39. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschr. Phys. **46**(4–5), 493–505 (1998)